Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.12785/amis/090258

Modeling and Resource Scheduling of Real-Time Unsplittable Data Transfers

Mustafa Müjdat Atanak[1,*], *Atakan Doğan*[2] *and Mustafa Bayram*[3]

[1] Department of Computer Engineering, Dumlupınar University, Kütahya, Turkey
[2] Department of Electrical and Electronics Engineering, Anadolu University, 26470 Eskişehir, Turkey
[3] Department of Mathematical Engineering, Yildiz Technical University, 34220 Istanbul, Turkey

Received: 29 Jun. 2014, Revised: 27 Sep. 2014, Accepted: 28 Sep. 2014
Published online: 1 Mar. 2015

Abstract: Real-time communication where the timely delivery of the data transfer requests needs to be guaranteed is essential for several applications. This work formally introduces the Real-Time Unsplittable Data Dissemination Problem (RTU/DDP), which is a generalization of the unsplittable flow problem. RTU/DDP problem is proved to be NP-hard. Therefore, heuristic approaches are required to acquire good solutions to the problem. The problem is divided into two sub-problems: path selection and request packing. Each of these sub-problems is formally defined and heuristic algorithms are proposed for both sub-problems. MinMin/FPF, Edge Disjoint MinMin/FPF, MinCon/FPF, and LFL-MinCon/FPF heuristics are proposed for the path selection subproblem. MNOFF and MOFF heuristics are introduced for the latter subproblem. The performances of these algorithms are compared with a genetic algorithm solution proposed in this study and a heuristic from the literature. The results and discussions of the comparisons among the performances of the proposed heuristics are presented.

Keywords: Real-time systems, modeling, QoS data management, data communication, scheduling

1 Introduction

Scientific and commercial applications are becoming more and more compute and data intensive in recent years. Such applications require the transfer of immense volume of data, reaching the order of terabytes. The topics of these applications vary from high-energy physics (EU-DataGrid [1]) to climate modeling (Earth System Grid [2]) to earthquakes (NEESit [3]). Data sources of these applications may vary from sensors in a hadron collider, to climatic and seismic data sensors located all around the globe, to satellite images. In most of these applications, data is of type write-once read-many, and data is required to be distributed to the researchers that are geographically distributed.

For example, Compact Muon Solenoid (CMS) Detector located at LHC (Large Hadron Collider) produces about 1 petabyte of read-only data every year. This data is stored at CERN and provided to other participating organizations (sites) upon request. As a result, multiple copies of a data item can coexist in the Grid system and the master copy is located at the storage elements at CERN [1]. A Grid System called LHC Computing Grid system (LCG) [4] is built to deliver the data to researchers. Ultra high-speed dedicated connections are set up to transfer the data to remote sites. The data dissemination problem in this context deals with the efficient and fast distribution of these data to requestor sites using the limited storage and network resources in the system.

The data dissemination problems focus on selecting the path over which the data will be delivered, and the amount of bandwidth that will be used for the connection. Best-effort service of Internet does not provide any guarantees regarding bandwidth, delay, loss rate, etc. While best-effort service is acceptable for traditional applications, such as FTP and email, it is intolerable for recent applications that impose real-time requirements on data transfers, such as internet telephony, video-conferencing, video on-demand, defence and surveillance [5], wireless sensor networks [6,7], and Grid systems [8]. Transferring the required data as early as possible is not adequate in these applications. The data transfers have to satisfy the QoS requirements in order to

* Corresponding author e-mail: mustafa.atanak@dpu.edu.tr

be regarded as successful. Providing an optimal real-time performance with these applications under the constraint of limited computing, storage, and network resources is very challenging task. Even offering some sort of real-time performance guarantees requires a high degree of coordination in the system.

In addition to meeting QoS requirements, data dissemination problems can aim to maximize the network utilization and improve the total throughput of the network. In this sense, simply routing a flow over a path that can meet the QoS requirements of flow is not good enough. The total resource allocation for a flow along its path, in relation to available resources needs to be taken into account. This mechanism is called admission control (or request packing). If this flow needs too many resources, it may be rejected even if the network has the capability to accept it. By doing so, the resources can be used by other flows which cost less. A related problem is fairness. Larger flows tend to need more resource while small flows need less. Thus, small flows always have a better chance to be accepted. In order to be fair, that larger flows can get a certain level of acceptance rate needs to be guaranteed.

In this study, the Real-Time Unsplittable Data Dissemination Problem (RTU/DDP), which is a generalization of the well-known Unsplittable Flow Problem, is introduced first. RTU/DDP tries to find out the routes that the data requests should take and the amount of bandwidth that should be assigned to each request so that the number of real-time requests that are delivered successfully is maximized. RTU/DDP is solved in two phases in which the first phase is Real-Time Path Selection Problem (RT/PSP) and the second one is Request Packing Problem (RPP). For the solution of RT/PSP, four different algorithms, namely MinMin/FPF, Edge Disjoint MinMin/FPF, MinCon/FPF, and LFL-MinCon/FPF, are proposed; for the solution of RPP, MNOFF and MOFF algorithms are introduced. By design, it is possible to mix any phase-1 algorithm with any phase-2 algorithm. Thus, eight different algorithms are possible to be used for solving RTU/DDP. According to the detailed simulation studies, MinCon/FPF and MOFF are the best combination in maximizing the number of real-time requests satisfied.

The rest of the paper is organized as follows: Section 2 summarizes the related work. Section 3 introduces the data dissemination model used in this study, and formally defines RTU/DDP. Section 4 presents the proposed solutions to RTU/DDP. Section 5 presents the simulation results and discussions. Finally, the last section provides conclusions and future directions of study.

2 Related work

Transferring data in an efficient and fast manner has been the topic of many studies in the literature. Data dissemination problems come in very different settings. In the real-time data dissemination problem that will be formally defined in this paper, the goal is to maximize the number of real-time data transfer requests satisfied. Similar problem definitions are encountered in the framework of QoS-based routing as well. In the literature, a variety of QoS-based routing problems have been defined and many QoS-based routing algorithms have been proposed. Most of them start from extending the ability of current best-effort routing algorithms. Thus, in this section, best-effort and QoS-based routing algorithms will be summarized.

2.1 Best-effort routing services

The solution space of best-effort routing problems is extremely large and an optimal solution cannot be found in a reasonable time. Instead, heuristics are employed to find good solutions [9]. Heuristics such as Dijkstra [10], Bellman-Ford [11,12] are used to solve the shortest distance problem. Nowadays, similar algorithms are frequently used in the state-of-the-art network systems. Most of the data dissemination algorithms in the literature are actually a modified or extended version of well-known shortest distance algorithms [13,14,15,16]. In IP-based networks, it is traditional to use protocols that include shortest distance algorithms.

Current Internet routing protocols such as RIP (Routing Information Protocol) [17], OSPF (Open Shortest Path First) [18], and BGP (Border Gateway Protocol) [19] are called best-effort routing protocols. They use only the shortest path to the destination. The shortest path may not be the path with shortest physical distance. The path with the least cost or fewest hop counts has been considered as the shortest path in literature as well. In the shortest distance algorithms, all the traffic is routed over the shortest paths. Even if some alternate paths exist, they are not used as long as they are not the shortest ones. This scheme may lead to the congestion of some links, while some other links are not fully used.

A popular example of a traditional routing algorithm used to find the shortest path with minimum cost is the Widest-Shortest Path (WSP) algorithm [20]. The WSP algorithm is an improvement over the Min-Hop algorithm that is used in OSPF protocol that selects a path with the minimum number of hop count. If there are several such paths to choose from, the one that allows the largest possible throughput to be reserved will be chosen. In Shortest-Widest Path (SWP) [20], the goal is to find a path that allows maximum throughput. In case of equality, the path with the minimum number of hops is selected. An extension to these algorithms is provided in [21], which selects a path with minimum hop count among all possible connected paths. If more than one path has minimum hop count, it selects a path with maximum route bandwidth where the route bandwidth of a path is the minimum available bandwidth of all links in the path. If more than one path has minimum hop count and

maximum route bandwidth, it selects a path with maximum total available bandwidth.

In a recent work, Jung et al. [22] evaluates and compares the performances of different data scheduling algorithms from the literature. In their study, the authors include a large set of algorithms, namely feasible path, minimum hop feasible path, widest/shortest feasible path, shortest/widest feasible path, shortest distance feasible path, dynamic alternative feasible path, OSPF like algorithms, k dynamic paths, k static paths, slotted sliding window, list sliding window, extended Bellman-Ford algorithms. The study concludes that minimum hop feasible path and dynamic alternative feasible paths are superior to the other algorithms in the sense of maximizing network utilization.

Other solutions to implement best-effort routing services that solve the data dissemination problem based on simulated annealing [23], tabu search [24], ant colony [25], flooding [26], and vector converting [27] techniques exist in the literature as well. But, the most popular alternative solutions are genetic algorithm based. Chang [28] and Munetomo [29,30] have applied the genetic algorithm to the shortest path routing problem. Both Chang and Munetomo use variable length chromosome with each chromosome consisting of nodes that are on the path from sender to receiver. The algorithm in [29] is practically feasible in a wired or wireless environment. It employs variable-length chromosomes for encoding the problem. Other researchers who also use genetic algorithm to solve the shortest path routing problem (or a variation of it) are Sinclair [31], Shimamoto [32], and Hamdan [33]. Yu et al. [34] proposes a genetic algorithm for QoS-based routing problem. They join maze algorithm to decoding process to solve the problem and claim that this method obtains better convergence and stability. There are several genetic algorithms that address different kinds of routing problems, such as multiple destination or multicasting routing problems [35,36,37].

2.2 QoS-based routing services

To provide QoS guarantees to flows, two tasks need to be accomplished. The first is to find a feasible path from source to destination, which can meet the QoS requirements; the second is to reserve the resources along the path. QoS-based routing does the first task, while the second one is handled by a resource reservation protocols. QoS-based routing and resource reservation are two different techniques that are used in conjunction for solving the data dissemination needs of a system. QoS-based routing itself cannot reserve resources, and resource reservation protocols are not supposed to find the feasible path.

2.2.1 Resource reservation protocols

Different from the best-effort services, QoS-based routing requires advance resource reservation mechanisms. A path is pre-determined and associated resources along the path (link bandwidth, buffer space, etc.) are reserved before the actual transmission. In other words, the path or connection between source and destination is setup first. When the transmission finishes, the path and associated resources are released.

The problem of providing QoS guarantees such as end-to-end delay bounds for applications has been the subject of many studies (e.g., [8,38,39,40,41]). In order to support end-to-end guaranteed service in the Internet, the IETF has defined the Integrated Services (Intserv) architecture [39]. Later in [40], the network element (router) behaviour required to deliver a guaranteed delay and bandwidth in the Internet were described. Furthermore, Resource Reservation Protocol (RSVP) [41] complements Intserv by enabling the resource reservations on the routers along the path. Based on [8, 38,39,40,41] and the related studies, as is common in related studies, this study assumes that the network allows the share of any link bandwidth to be reserved for the transmission of data items with deadlines.

2.2.2 QoS-based routing

PNNI (Private Network-Network Interface) [42] and QOSPF (QoS routing extensions to OSPF) [43] are both based on the link-state algorithms that support QoS. They both require that every node try to acquire a map of the underlying network topology and its available resources via flooding. PNNI is used in ATM networks to route connections based on the network state and topology information. It is the only standardized QoS-based routing protocol. To support QoS in QOSPF, routers not only advertise topology information but also network resource information. The network resource information includes both router and link resources. Links that do not satisfy the QoS requirement are excluded from the path computation. The path computation algorithm in QOSPF pre-computes a widest-shortest path, which is a minimum hop count path with maximum bandwidth. Its computational complexity is comparable to that of Bellman-Ford shortest-path algorithm.

Several variations of QoS-based routing problems and some proposed solutions to these problems could be summarized as follows:

Bandwidth-bounded routing: For the incoming data transfer requests, the required bandwidth values are determined. Paths with sufficient bandwidths are considered as feasible solutions. Several solutions have been proposed to this problem [44,45,46].

Bandwidth-bounded, delay-optimized routing: This problem can be either solved as a widest-shortest path problem or a shortest-widest path problem [47].

In [48], Grimmell et al. formulated a dynamic quickest path problem, which deals with the transmission of a message from a source to a destination with the minimum end-to-end delay over a network with propagation delays and dynamic bandwidth constrains on the links. Yang et al. [49] computes the delay-weighted capacity for each ingress-egress pair. The authors propose an algorithm that avoids the use of the critical links by assigning large weights to them. Critical links are defined as those links whose inclusion in a path will cause the delay-weighted capacity of several ingress-egress pairs to decrease.

Bandwidth-bounded, cost-bounded routing: Solutions to this problem typically map the cost or the bandwidth to a bounded integer value, and solve the problem in polynomial time using an Extended Bellman-Ford (EBF) or Extended Dijkstra Shortest Path (EDSP) algorithm [50].

Multi-constrained routing: The objective of multi-constrained routing is to simultaneously satisfy a set of constraints [14,45]. Korkmaz et al. [14] proposes a heuristic approach for the multi-constrained optimal path problem (MCOP), which optimizes a non-linear function (for feasibility) and a primary function (for optimality).

3 Problem formulation

A networked system is modeled by an undirected graph $G = (V,E)$, where $V = \{v_1,...,v_n\}$ defines the heterogenous machines and $E = \{e_1,...,e_m\}$ denotes the links each of which connects any two machines of the system. The machines can be storage elements with limited storage space as well as network routers (or switches). Each $e_i \in E$ is associated with a bandwidth value $c_i > 0$ and a delay value $d_i \geq 0$.

$R = \{r_1,...,r_\lambda\}$ denotes a set of real-time data transfer requests. Each request $r_i \in R$ is modeled by a quadruple $< s_i,t_i,f_i,\delta_i >$ in which s_i is the source machine, t_i is the destination machine, f_i is the requested data item, and $\delta_i > 0$ is the deadline of request r_i.

$P_i = \{p_1,...,p_{l_i}\}$ defines a set of paths for request $r_i \in R$, where $p_j \in P_i$ is a simple path which connects machines s_i and t_i; $l_i > 0$ is the number of such paths. Furthermore, $P_i(e_k) = \{p_j : p_j \in P_i \text{ and } e_k \in E\}$ denotes a set of paths $p_j \in P_i$ $(j \leq l_i)$ each of which includes link $e_k \in E$ for request $r_i \in R$.

Definition 3.1. The bandwidth demand of request $r_i \in R$ on path $p_j \in P_i$, which is denoted by π_{ij}, is the minimum bandwidth value that guarantees the timely delivery of r_i at its destination.

$$\pi_{ij} = \frac{|f_i|}{\delta_i - \sum_{e_k \in p_j} d_k} \tag{1}$$

where $|f_i|$ denotes the data item size.

Definition 3.2. A path $p_j \in P_i$ is feasible for $r_i \in R$ if and only if $\beta_{ij} \geq \pi_{ij}$, where β_{ij} is the bottleneck bandwidth of path p_j and equal to the minimum of available bandwidth values of all the links that constitute the path.

Definition 3.3. A request $r_i \in R$ is satisfied if there exists at least one feasible path in P_i.

Definition 3.4. The satisfiability of a set of real-time data transfer requests is the number of satisfied requests in R by means of a scheduling algorithm.

In the system, there is a *centralized data dissemination scheduler* in charge of making all real-time data scheduling decisions for all the data transfer requests in R submitted by the running applications. Furthermore, the scheduler is capable of issuing reservation requests to the respective system components in order for the data transfers to take place as scheduled. As a result, when a data item f_i with deadline δ_i needs to be moved from a source s_i to its destination t_i storage element, the scheduler calls for a data dissemination algorithm that computes a path and bandwidth value, and one or more system components collectively reserve the computed bandwidth value on all links along the path until the end of the transmission.

For the best-effort data dissemination problems studied in the literature, a variety of goals are attained. Some best-effort goal examples include the completion of data transfer requests as early as possible, minimizing a cost function associated with the data transfers or maximizing over all link utilization. Since this work focuses on the real-time data transfers, the goal differs from the best-effort problems, and Definition 3.5 formally gives it.

Definition 3.5. Given a networked system $G = (V,E)$ and a set of real-time data transfer requests R, the Real-Time Unsplittable Data Dissemination Problem (RTU/DDP) seeks to maximize the satisfiability of R.

$$\begin{aligned} RTU/DDP : max & \sum_{r_i \in R} \sum_{p_j \in P_i} x_{ij} \\ & \sum_{p_j \in P_i} x_{ij} \leq 1, \forall\, r_i \in R \\ & \sum_{r_i \in R} \sum_{p_j \in P_i(e_k)} \pi_{ij} x_{ij} \leq c_k, \forall\, e_k \in E \\ & x_{ij} \in \{0,1\}, \forall\, r_i \in R \text{ and } p_j \in P_i \end{aligned} \tag{2}$$

where x_{ij} is 1 if request $r_i \in R$ is transferred over $p_j \in P_i$, and 0 otherwise.

In RTU/DDP, the objective function is to maximize the number of satisfied requests. For any request $r_i \in R$, in the first constraint, the number of paths that are used to make it satisfiable must be less than or equal to one. The second constraint for all links $e_k \in E$ requires that their capacity must not be violated due to all scheduled data transfers.

Theorem 3.1. RTU/DDP is NP-hard.

Proof. The Unsplittable Flow Problem (UFP) known to be NP-hard [51] can be reduced to the RTU/DDP in polynomial time. For the UFP, different from the RTU/DDP, each request $r_i \in R$ is modeled with

$< s_i, t_i, \pi_i, \omega_i >$ quadruple in which $\pi_i > 0$ and $\omega_i > 0$ show the demand and profit of $r_i \in R$, respectively. Note that UFP includes neither link delays nor request deadlines.

$$UFP: max \sum_{r_i \in R} \sum_{p_j \in P_i} \omega_{ij} x_{ij}$$
$$\sum_{p_j \in P_i} x_{ij} \leq 1, \forall\, r_i \in R$$
$$\sum_{r_i \in R} \sum_{p_j \in P_i(e_k)} \pi_i x_{ij} \leq c_k, \forall\, e_k \in E$$
$$x_{ij} \in \{0,1\}, \forall\, r_i \in R \text{ and } p_j \in P_i \quad (3)$$

The UFP and RTU/DDP becomes equivalent if (1) $\omega_i = 1$ for all $r_i \in R$, and (2) $\pi_i = \pi_{ij} = |f_i|/\delta_i$ for all $r_i \in R$, where $d_i = 0$ for all $e_i \in E$ is assumed.

Based on Theorem 3.1, an optimal solution to RTU/DDP cannot be found in a polynomial time, unless P = NP. Thus, heuristic approaches, e.g., [5,52], are adopted to find good solutions to RTU/DDP. In this study, in order to solve RTU/DDP, a two phase real-time data scheduling approach is proposed in the subsequent sections.

4 Real-time unsplittable data dissemination

In this study, RTU/DDP is split into two sub-problems, namely Real-Time Path Selection Problem (RT/PSP) and Request Packing Problem (RPP), whose solutions are sought in two separate phases. While RTU/DDP is solved, in the first phase, a heuristic algorithm produces a solution to RT/RSP, in which a single feasible path for each request is included, if possible. Thus, after the first phase, there are as many feasible paths as the number of requests, or less. However, having a feasible path for each request does not usually imply that all requests can be simultaneously scheduled along their respective feasible paths because of the link capacity constraints. In such a case, a small number of requests must be rejected for the sake of the remaining ones, which now can be satisfied along their pre-computed paths. In this study, deciding about which requests should stay and which ones should be rejected is handled in the second phase. Thus, another heuristic algorithm takes all the paths generated in the first phase as an input and produces a solution to RPP, in which a subset of input paths with the minimum cardinality is rejected. In the following sections, several algorithms for the solution of RT/PSP and RPP are introduced.

4.1 Real-time path selection

The first phase, in which a feasible path is determined for all requests, is Real-Time Path Selection Problem that is formally defined as follows.

Definition 4.1. The congestion, which is denoted by ζ, is the maximum number of data transfers routed over any link in the network when a feasible path is chosen for every request.

$$\zeta = \max_{e_k \in E} \sum_{r_i \in R} \sum_{p_j \in P_i^*(e_k)} x_{ij} \quad (4)$$

where $P_i^*(e_k)$ is the set of feasible paths each of which includes edge e_k for request $r_i \in R$.

Definition 4.2. Given a networked system $G = (V,E)$ and a set of real-time data transfer requests R, the Real-Time Path Selection Problem (RT/PSP) seeks to minimize the congestion.

$$RT/PSP: min\ \zeta$$
$$\sum_{p_j \in P_i^*} x_{ij} \leq 1, \forall\, r_i \in R$$
$$x_{ij} \in \{0,1\}, \forall\, r_i \in R \text{ and } p_j \in P_i \quad (5)$$

Theorem 4.1. RT/PSP is NP-hard.

Proof. RT/PSP is equivalent to Congestion Minimization Problem, which is known to be NP-hard [53].

For the solution of RT/PSP, in this study, four different algorithms are proposed. These algorithms adopt different heuristics with different time complexities as explained below.

4.1.1 Minimum hop minimum delay feasible path first algorithm

The Minimum Hop Minimum Delay Feasible Path First (MinMin/FPF) algorithm is based on the heuristic that the congestion is minimized if every request gets routed over a *feasible path* with minimum number of hops incurring a small path delay. Note that a minimum hop feasible path includes the least number of links to satisfy that request. As a result, choosing a minimum hop feasible path for every request leads to minimizing the total number of network links that will be occupied by all request, which is likely to minimize the congestion. Fig. 1 shows the Minimum Hop Minimum Delay Feasible Path Heuristic (MinMin/FPH). Using MinMin/FPH algorithm, MinMin/FPF algorithm finds the feasible paths for all requests, which is shown in Fig. 2.

MinMin/FPH is based on the Bellman-Ford shortest path algorithm, and it is inspired by [54]. In Fig. 1, each vertex v is associated with a hop count value ($v.hopcount$) and a predecessor vertex ($v.pred$), where $v.hopcount$ is the number of hops from *source* to vertex v; $v.pred$ is the predecessor vertex of vertex v in the minimum hop minimum delay path from *source* to vertex v. Each edge e is associated with a source vertex ($e.source$), a destination vertex ($e.dest$), a delay value ($e.delay$), and a weight value ($e.weight$). During the initialization, $v.hopcount$ is set to infinity if the vertex is not the source of request

```
MinMin/FPH algorithm
  //Input: G(V,E), request r
  //Output: path: minimum hop minimum delay feasible path
  //Initialization
  source = r.source;
  for each vertex v ∈ V
    if v is source then v.hopcount = 0;
    else v.hopcount = ∞; end if
    v.pred =null;
  end for
  //Main loop
  for each vertex v ∈ V
    for each edge e ∈ E
      if e.dest.hopcount > e.source.hopcount + e.weight
      then
        e.dest.hopcount = e.source.hopcount + e.weight;
        e.dest.pred = e.source;
      else if e.dest.hopcount = e.source.hopcount + e.weight
      then
        Calculate PathDelay1 from source to e.dest;
        Calculate PathDelay2 from source to e.source;
        if PathDelay1 > PathDelay2 + e.delay then
          e.dest.pred = e.source;
        end if
      end if
    end for
  end for
  //Construct min-hop min-delay path
  v = r.destination;
  repeat
    Add the edge between v.pred and v into path;
    v = v.pred;
  until v ≠null;
  if path is feasible then return path;
  else return null; end if
end algorithm
```

Fig. 1: MinMin/FPH algorithm

(*r.source*), in which case *v.hopcount* is set to zero. In addition, since MinMin/FPH takes $G = (V,E)$ as an input, for every edge, the values of *e.source*, *e.dest*, and *e.delay* are already available to the algorithm. On the other hand, *e.weight* is set to one by MinMin/FPF during its initialization phase.

In the main loop, the algorithm tries to find the minimum hop count path between the source (*r.source*) and destination (*r.destination*) vertices since *e.weight* is one for all edges. If more than one path has the same hop count, the path with minimum delay is favored. Once the main loop ends, *v.hopcount* and *v.pred* have been determined for all vertices in the network. For the request of interest, the minimum hop minimum delay path is constructed by following *v.pred* from destination to source vertex. Finally, the path found by the algorithm is checked for feasibility. If the path is not feasible, an empty path is returned; otherwise, the path found is

```
MinMin/FPF algorithm
  //Input: G(V,E), set of requests R
  //Output: P: min-hop min-delay feasible paths for requests
  //Initialization
  for each edge e ∈ E
    e.weight = 1;
  end for
  //Main loop
  for each request r ∈ R
    Add MinMin/FPH(G,r) into set P;
  end for
end algorithm
```

Fig. 2: MinMin/FPF algorithm

returned. In Fig. 2, MinMin/FPF calls MinMin/FPH once for each request in the main loop. Then, it inserts the path found by MinMin/FPH into set *P*.

The time complexity of minimum hop Bellman Ford algorithm is $O(|V||E|)$. MinMin/FPH algorithm additionally calculates path delays in each step, which takes $|E|$ steps in the worst case. Thus, MinMin/FPH runs in $O(|V||E|^2)$ for a single request. Since MinMin/FPH should be run for each request $r_i \in R$, the time complexity of MinMin/FPF becomes $O(|V||E|^2|R|)$.

4.1.2 Edge disjoint minimum hop minimum delay FPF algorithm

MinMin/FPF gives higher priority to the paths with fewer hops. However, the selected paths by MinMin/FPF may still use some of the links more than the others. That is, while some links are under-utilized, some others may be exhausted. Yet, such an unbalanced use of links will cause the congestion increase

The Edge Disjoint Minimum Hop Minimum Delay FPF (Edge Disjoint MinMin/FPF) algorithm adopts the heuristic that the congestion is minimized if every request gets routed over a *feasible path* with minimum number of hops incurring a small path delay that are edge disjoint. If Edge Disjoint MinMin/FPF finds an edge disjoint path for every request, all requests are guaranteed to be satisfied in the second phase, in which case the congestion becomes one. Unfortunately, the congestion value of one is usually impossible to attain. Yet, Edge Disjoint MinMin/FPF keeps the congestion under control by selecting edge-disjoint paths as much as possible.

As shown in Fig. 3, Edge Disjoint MinMin/FPF sorts all requests in increasing order based on their bandwidth demands during initialization and later considers them for scheduling in the sorted order. In addition, the algorithm sets all edge weights to one, which makes MinMin/FPH return a minimum hop and minimum delay feasible path for the respective request, if possible. Inside the main loop, Edge Disjoint MinMin/FPF tries to find out edge disjoint paths for the requests with the help of

Edge disjoint MinMin/FPF algorithm
//Input: $G(V,E)$, set of requests R
//Output: P: edge disjoint min. hop min. delay feasible paths
//Initialization
Calculate $\pi_i = |f_i| \div \delta_i$ for each request $r_i \in R$;
Sort requests in increasing order wrt π_i;
for each edge $e \in E$
$e.weight = 1$;
end for
//Main loop
repeat
Restore G to include all vertices and edges in the network;
repeat
Pick the first *unprocessed* request r in the sorted order;
if ($path$ =MinMin/FPH(G, r)) $\neq$ **null then**
Add $path$ into set P;
Mark r as *processed*;
$G = G - \{e : e \in path\}$;
else
if G includes all vertices and edges **then**
Mark r as *processed*;
end if
end if
until Graph G becomes disconnected;
until All requests are processed;
end algorithm

Fig. 3: Edge disjoint MinMin/FPF algorithm

MinMin/FPH. That is, when MinMin/FPH finds out a path for a request, a reduced graph is obtained by temporarily deleting all links used by this path. For the next request, MinMin/FPH will be looking for a path in this reduced graph, which ensures that the paths returned by MinMin/FPH for these two requests are edge disjoint. While deleting the links from the graph, at some point, the graph becomes disconnected and the algorithm restores the graph and continues as before. As a result, it is more likely that not all paths returned by Edge Disjoint MinMin/FPF are edge disjoint.

In Edge Disjoint MinMin/FPF algorithm, the worst-case scenario occurs when only one request can be marked as *processed* by inner repeat loop until the graph gets disconnected. Yet, MinMin/FPH is called for all currently unprocessed requests by inner repeat loop. Thus, MinMin/FPH needs to be called $O(|R|^2)$ times, which leads to the time complexity of $O(|V||E|^2|R|^2)$.

4.1.3 Minimum contention feasible path first algorithm

Both MinMin/FPF and Edge Disjoint MinMin/FPF assume that each edge has a unity weight value as far as the congestion is concerned, and they do not consider the link bandwidths while computing paths. On the other hand, the network is composed of links with different bandwidth values and the admissible congestion values

MinCon/FPF algorithm
//Input: $G(V,E)$, set of requests R
//Output: P: min. contention feasible paths for requests
//Initialization
Find $AverageBandwidth$ using all $e.bandwidth$ values;
for each edge $e \in E$
$e.weight = 1$;
end for
//Main loop
for each request $r \in R$
Add $path$ =MinMin/FPH(G,r) into set P;
for each edge $e \in path$
$e.weight = e.weight +$
$Alpha \times AverageBandwidth \div e.bandwidth$;
end for
end for
end algorithm

Fig. 4: MinCon/FPF algorithm

for different links can be different. In order to take this fact into account, the links with low bandwidth values can be assigned higher weight values in commensurate to their bandwidths as compared to the ones with high bandwidth values. Then, during the computation of minimum weight paths for requests by a shortest-path algorithm, such a weight assignment will favor the links with high bandwidth and minimize the use of the links with low bandwidth.

The Minimum Contention Feasible Path First (MinCon/FPF) algorithm, as shown in Fig. 4, is very similar to MinMin/FPF. The main difference between these two algorithms is that the former one dynamically adjusts the link weights in order to reflect the current link congestion more accurately. This dynamic weight adjustment of links is carried out as follows. First, $AverageBandwidth$ value is computed over all links available in the network. When a link is used for delivering a data item, its weight is increased by some amount based on the ratio $AverageBandwidth/e.bandwidth$. If the link has relatively low/high bandwidth value, $AverageBandwidth/e.bandwidth$ is greater/less than one. Thus, such a link weight adjustment puts more weight on low-bandwidth links so that they will have less chance of being selected by MinMin/FPH for the next request. Eventually, low-bandwidth links will have lower congestion values as compared to high-bandwidth links. The parameter $Alpha > 0$ is further used to further increase or decrease the impact of $AverageBandwidth/e.bandwidth$ ratio during the weight computation. The time complexity of MinCon/FPF is $O(|V||E|^2|R|)$.

4.1.4 Link-flow limited minimum contention feasible path first algorithm

Both Edge Disjoint MinMin/FPF and MinCon/FPF try to minimize link congestion by adopting very different approaches: the former one relies on the edge disjoint paths, while the latter one is based on the dynamic weight adjustment. The Link-Flow Limited Minimum Contention Feasible Path First (LFL-MinCon/FPF) algorithm, which is shown in Fig. 5, aims at unifying these two approaches to further minimize the congestion.

As shown in Fig. 5, LFL-MinCon/FPF algorithm works very similar to Edge Disjoint MinMin/FPF. The main differences between these two algorithms are as follows. (1) When Edge Disjoint MinMin/FPF finds a path, it temporarily deletes all the links on this path before the start of the next iteration for an unprocessed request. On the other hand, LFL-MinCon/FPF first re-computes the link weights on the path, which is similar to MinCon/FPF. Then, if the new weight value of any edge is greater than a loop count-adjusted threshold value ($AllowableLinkWeight \times LoopCount$), only this link (not all the links on the path found) is removed from the network. (2) Edge Disjoint MinMin/FPF always returns minimum hop and minimum delay paths for requests. However, this is not necessarily true for LFL-MinCon/FPF due to the dynamic weight computation.

In Fig. 5, *AllowableLinkWeight* is a threshold parameter, which is received as an input parameter by the algorithm. During operation of the algorithm, the network may get disconnected before marking all requests as processed in iteration *LoopCount*. When this happens, the graph is first restored. Then, a link will be deleted from the graph only if its weight now exceeds $AllowableLinkWeight \times (LoopCount + 1)$. The time complexity of LFL-MinCon/FPF is $O(|V||E|^2|R|^2)$, which is the same as that of Edge Disjoint MinMin/FPF.

LFL-MinCon/FPF algorithm
//Input: $G(V,E)$, request r
//Output: *path*: link flow limited minimum contention feasible paths for requests
//Initialization
Calculate $\pi_i = |f_i| \div \delta_i$ for each request $r_i \in R$;
Sort requests in increasing order wrt π_i;
Find *AverageBandwidth* using all *e.bandwidth* values;
for each edge $e \in E$
$e.weight = 1$;
end for
//Main loop
$LoopCount = 1$;
repeat
Restore G to include all vertices and edges in the network;
repeat
Pick the first *unprocessed* request r in the sorted order;
if ($path$ =MinMin/FPH(G, r)) $\neq$ **null then**
Add *path* into set P;
Mark r as *processed*;
for each edge $e \in E$
$e.weight = e.weight +$
$Alpha \times AverageBandwidth \div e.bandwidth$;
if $e.weight > AllowableLinkWeight \times LoopCount$ **then**
$G = G - \{e : e \in path\}$;
end if
end for
else
if G includes all vertices and edges **then**
Mark r as *processed*;
end if
end if
until Graph G becomes disconnected;
$LoopCount = LoopCount + 1$;
until All requests are processed;
end algorithm

Fig. 5: LFL-MinCon/FPF algorithm

4.2 Request packing

Once one of the four algorithms is used to determine a path for each request, it is the responsibility of the request packing algorithm to maximize the number of satisfied requests. The request packing problem is formally defined below.

Definition 4.3. Given an undirected graph $G = (V,E)$ and a set of real-time data transfer requests R, in which each request $r_i \in R$ is modelled with $< p_i, \pi_{ij} >$ tuple, the Request Packing Problem (RPP) seeks to maximize the number of satisfiable requests.

$$RPP : max \sum_{r_i \in R} x_i$$
$$\sum_{r_i \in R} \sum_{p_j \in P^{\dagger}(e_k)} \pi_{ij} x_i \leq c_k, \forall\, e_k \in E$$
$$x_{ij} \in \{0,1\}, \forall\, r_i \in R \text{ and } p_j \in P_i \qquad (6)$$

where x_i is 1 if request $r_i \in R$ is satisfied, and 0 otherwise; $P^{\dagger}$ is a set of feasible paths for request $r_i \in R$ and $P^{\dagger}(e_k)$ is the set of feasible paths that include link $e_k \in E$.

Theorem 4.2. RPP is NP-hard.

Proof. Multidimensional 0-1 Knapsack Problem (MKP) [55] known to be NP-hard can be reduced to a RPP in polynomial time.

$$MKP : max \sum_{i=1}^{n} \omega_i x_i$$
$$\sum_{i=1}^{n} a_{ij} x_i \leq c_j, 1 \leq j \leq m$$
$$x_i \in \{0,1\}, 1 \leq i \leq n \qquad (7)$$

where n is the number of items, m is the number of knapsacks with capacity $c_j > 0$, $\omega_i > 0$ is the profit of

including the ith item, x_i is 1 if the ith item is included into a knapsack and 0 otherwise, and a_{ij} ($0 \leq a_{ij} \leq c_j$ for all $1 \leq j \leq m$) is the resource consumed by the ith item in the jth knapsack. The MKP and RPP becomes equivalent if

– Let n and m be equal to the number of requests and edges, respectively,
– $\omega_i = 1$ for $1 \leq i \leq n$, and
– $a_{ij} = \pi_{ij} = |f_i|/\delta_i$ for all $r_i \in R$, where $d_i = 0$ for all $e_i \in E$ is assumed.

MKP is a well-known integer-programming problem. Exact algorithms based on branch and bound and dynamic programming are proposed, but they work in modest size problems only. Thus, in this study, two heuristic algorithms are proposed to solve RPP. Both algorithms are based on the concept of contention graph.

Definition 4.4. The bottleneck link is a link whose bandwidth capacity will be exceeded by the total bandwidth demand of one or more requests if these requests were scheduled to use the link.

Definition 4.5. The contention graph is a bipartite graph whose vertices correspond to all requests and all bottleneck links, and whose edges connect a request vertex to a bottleneck link vertex provided that the path of request includes this bottleneck link.

4.2.1 Maximum number of outgoing flows first

The Maximum Number of Outgoing Flows First (MNOFF) algorithm is shown in Fig. 6. Initially, all requests are satisfiable. In its main loop, MNOFF first produces a contention graph based on the paths chosen by the respective path selection algorithm by following Definition 4.4 and 4.5. Then, in each iteration, MNOFF drops the request with the maximum number of outgoing flows (the vertex with the maximum number of edges in the contention graph) until no edge is left in the graph. In the case of equality, MNOFF picks the request with the highest bandwidth demand. Once a request is deemed to be unsatisfiable, MNOFF updates the contention graph and continues until all edges are removed.

During the operation of MNOFF, the contention graph is updated for every request deemed to be unsatisfiable until all edges are removed from the graph. The rationale behind such an update and the update scheme are as follows. In the contention graph, an edge indicates that one or more than one request will not be satisfied. If a request is considered to be unsatisfiable, the contention graph is required to be updated to reflect the drop of this request. That is, the bandwidth demand of dropped requests should be subtracted from the used bandwidths of the related bottleneck links and the results should be compared with the respective link bandwidth capacities. If a bottleneck link is found be not bottleneck anymore, the vertex for this link and all the connections made to the vertex should be excluded from the graph. Thus, as long as there are edges in the contention graph, there exist unsatisfiable requests.

```
MNOFF/MOFF algorithm
  //Input: G(V,E), set of requests R, set of paths P
  //Output: list of satisfied requests
  //Initialization
  for each request r ∈ R
    r.status = satisfiable;
  end for
  //Main loop
  Form contention graph;
  repeat
    if MNOFF then
      r : request with maximum number of outgoing flow;
    else
      r : request with maximum outgoing flow;
    end if
    Let r.status = unsatisfiable;
    Update contention graph;
  until Contention graph has an edge;
end algorithm
```

Fig. 6: MNOFF/MOFF algorithm

Finding the request with maximum number of outgoing flow is $O(|E||R|)$ and updating the contention graph takes $O(|E||R|)$ in the worst case. Since each iteration marks one request as unsatisfiable, the loop repeats $|R|$ times. Therefore, the time complexity of MNOFF algorithm is $O(|E||R|^2)$.

4.2.2 Maximum outgoing flows first

The Maximum Outgoing Flows First (MOFF) algorithm is depicted in Fig. 6. The only difference between MOFF and MNOFF is the criterion used to choose unsatisfiable requests. That is, the criterion of MNOFF is the number of edges leaving a request vertex, while that of MOFF is the number of edges leaving a request vertex times the demand of this request, which is referred as the maximum outgoing flow. The time complexity of MOFF algorithm is $O(|E||R|^2)$, which is the same as that of MNOFF algorithm.

5 Experimental results

In order to evaluate the performance of the algorithms, a simulation program that can be used to emulate the execution of randomly created data transfer requests on a simulated network was developed. The simulator was written in C++ programming language. The performances of the algorithms are tested against a genetic algorithm (GA) solution of the RTU/DDP and a popular best effort

algorithm, namely Full Path Heuristic (FPH) algorithm, that can solve RTU/DDP from the literature [5].

Genetic algorithm based solutions are employed in the solution of the data dissemination problems as well as most of the optimization problems. Guided search mechanisms in the genetic algorithms may guide the algorithm designers to engineer algorithms with better performances. Genetic algorithms include four main components: candidate solutions are encoded in a binary string or matrix and they are artificially evolved to toward better solutions. In order to produce the next generation of solutions, some of the solutions of the initial population are selected via a selection operation. Then, during the reproduction process, next generation population is produced from the initial population through two possible genetic operators: cross over and mutation. The process is repeated until a termination condition is reached.

In the GA solution proposed in this study, the algorithm starts by finding PATH_DIVERSITY possible paths for each request using a k-shortest path algorithm. In the final solution, at most one path can be chosen for a single request. Two dimensional NUMBER_OF_REQUESTS by PATH_DIVERSITY solution matrix SM is defined as follows: SM[i][j] is equal to 1 if path j is chosen for request i, and 0 otherwise. Each row in SM matrix designates the selected path information for a single request, and since the flows are unsplittable, at most one entry of SM matrix can be 1 for a single row. A population is a set of solution matrices. RTU/DDP becomes finding the fittest solution matrix.

Roulette wheel implementation is used as the selection operation in the solution. This operation ensures that fitter solutions receive a higher probability in entering the genetic operators of the reproduction stage. In proposed cross over operation, two solution matrices are chosen by the selection operation. Two different cross over sites are chosen from 0 to NUMBER_OF_REQUESTS. Two new solutions are generated by swapping the rows of chosen solution matrices that lies between the cross sites. Mutation operation is defined as follows: a solution matrix is chosen by the selection operation. Two random numbers are created: one number selects a request and the second number selects a path. A new solution is generated by assigning the selected path to selected request. If the solution is valid, the fitness value is equal to number of ones in solution matrix, and zero otherwise. The fitness values of the new solutions are evaluated. The solutions with lowest fitness values are deleted from the population.

The performances of the algorithms are tested in three different network settings. First experiment set uses a network based on the topology of the GÈANT network (as known in April 2004). GÈANT is a pan-European multi-gigabit data communications network, reserved specifically for research and education use. The network is detailed in [56] and has 33 nodes and 44 links. LCG network (as of 2008) is used as to identify the underlying topology in the second experiment. LCG network (as of 2008) [4] has 151 nodes and 164 links. As the third set of experiments, a random topology, which is called RT network (Random Topology) in this study, of 100 nodes a 400 links is randomly created. The links in RT network have bandwidths uniformly distributed in 20% neighbourhood of AVE_BANDWIDTH (between 0.8 $\times$ AVE_BANDWIDTH and 1.2 $\times$ AVE_BANDWIDTH) and delays uniformly distributed in 20% neighbourhood of AVE_DELAY.

During the simulations, all data transfer requests are assumed to come in to the system at time zero. After generating the network topology, NUMBER_OF_REQUESTS requests are generated with the following parameters: AVE_REQUEST_SIZE and AVE_DEADLINE. Table 1 presents the values of these parameters in the base simulation study. Request sizes and deadline values of a request reside within 20% neighbourhood of the two parameters.

In each experiment set, the base studies are performed first. Then, individual parameters are varied to analyze the effect of the parameter on the real-time performance of the algorithms.

5.1 Results using GÈANT network

As part of the tests with GÈANT network, a base set of results was established with parameter values of NUMBER_OF_REQUESTS=1000, AVE_REQUEST_SIZE=10 Gbit and AVE_DEADLINE=200 sec. The base results are shown in Table 1. Each data in Table 1 denotes the average satisfiability (the percentage of number of satisfied requests to total number of requests) in one hundred simulation runs.

Table 1: Base results of test set using GÈANT network.

Path Selection	Request Packing	Base Results
MinMin/FPF	MNOFF	52.65
	MOFF	54.16
ED MinMin/FPF	MNOFF	52.82
	MOFF	52.91
MinCon/FPF	MNOFF	54.11
	MOFF	54.28
LFL-MinCon/FPF	MNOFF	46.08
	MOFF	46.08
GA		52.97
FPF		50.76

As it can be seen from the results in Table 1, the best satisfiability results are obtained if MinCon/FPF is used as the underlying path selection algorithm and MOFF is selected as the underlying request packing algorithm. For all path selection algorithms, MOFF request packing algorithm performs at least as good as MNOFF algorithm.

LFL-MinCon/FPF algorithm does not yield good real-time performances.

As shown in Figure 7, if the number of requests that is submitted to the system is increased, the real-time performance results decrease. In all cases, (MinCon/FPF, MOFF) is sharing the lead with (Edge Disjoint MinMin/FPF, MOFF) tuple. LFL-MinCon/FPF performs the worst

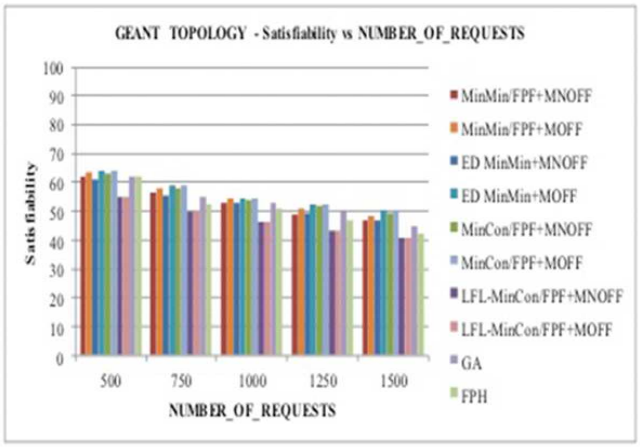

Fig. 7: Effect of varying number of requests on real-time performance in GÈANT network

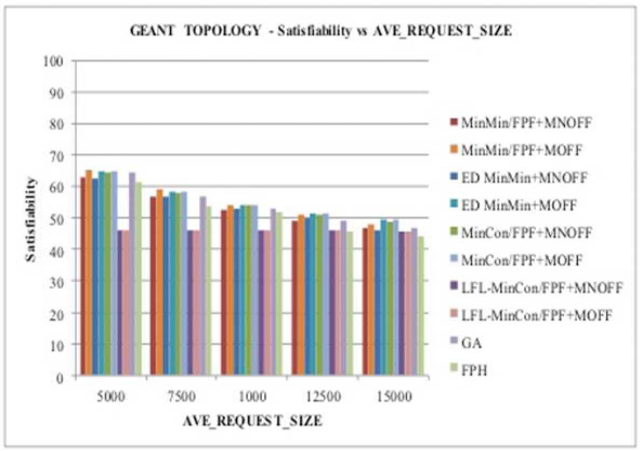

Fig. 8: Effect of varying average request size on real-time performance in GÈANT network

Figure 8 shows the effect of varying the average request size. Increasing the average request size gradually from 5 Gbit to 15 Gbit, decreases the average satisfiability results. In almost all cases, (MinCon/FPF, MOFF) is sharing the lead with (Edge Disjoint MinMin/FPF, MOFF) tuple. LFL-MinCon/FPF yields the worst results.

5.2 Results using LCG network

The number of nodes LCG network is 151 and the number of links is 164. The number of nodes and links in GÈANT network was 33 and 44, respectively. Clearly, LCG network is a bigger network than GÈANT. Furthermore, the number of links per number of nodes ratio is smaller. Immediate consequence of this is the overall decrease in performance results which is evident in Table 2.

Table 2: Base results of test set using LCG network.

Path Selection	Request Packing	Base Results
MinMin/FPF	MNOFF	33.23
	MOFF	34.53
ED MinMin/FPF	MNOFF	33.15
	MOFF	34.52
MinCon/FPF	MNOFF	35.25
	MOFF	35.92
LFL-MinCon/FPF	MNOFF	31.90
	MOFF	31.89
GA		33.98
FPF		31.61

A comprehensive analysis of the results in Table 2, together with the results shown in Figures 9 and 10 shows that the best real-time performance results are still obtained with (MinCon/FPF, MOFF) tuple.

5.3 Results using random networks

In RT network, 400 links are randomly placed between 100 nodes. Link bandwidths and delays are chosen within 20% neighborhood of the average values of AVE_BANDWIDTH=500 Mbit/sec and AVE_DELAY=5 msec. Since there exists a large number of links per number of nodes ratio, average performance results are larger than both GÈANT and LCG networks.

Results presented in Table 3 and Figures 11-12 are consistent with the results of GÈANT and LCG networks. The best performance results are obtained by (MinCon/FPF, MOFF) tuple. In Figures 11-12, LFL-MinCon/FPF performs similar results as MinCon/FPF algorithm. However, the time complexity of MinCon/FPF is much smaller than LFL-MinCon/FPF and should be chosen as the preferred path selection algorithm.

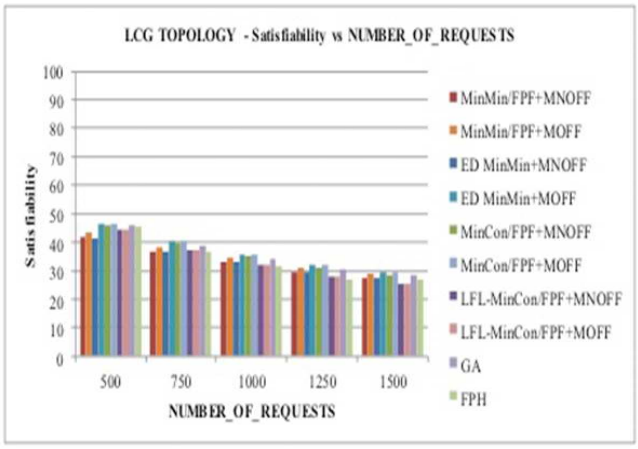

Fig. 9: Effect of varying number of requests on real-time performance in LCG network

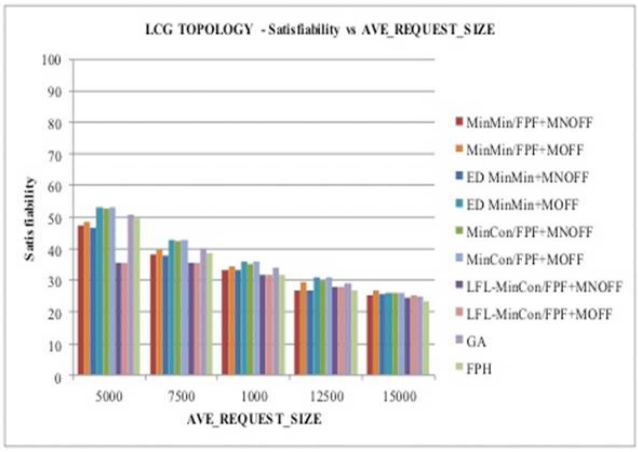

Fig. 10: Effect of varying average request size on real-time performance in LCG network

6 Conclusions

This work presented a real time data dissemination model and formally introduced a data dissemination problem which is referred as Real-Time Unsplittable Data Dissemination Problem (RTU/DDP). The problem is divided into two subproblems: path selection and request packing. Path selection algorithm tries to find possible paths for each request. Request packing algorithm decides which requests should be satisfied to maximize the number of satisfiable requests. MinMin/FPF, Edge Disjoint MinMin/FPF, MinCon/FPF, and LFL-MinCon/FPF algorithms are proposed for possible path selection algorithms. MNOFF and MOFF algorithms are proposed for possible request packing algorithms.

Table 3: Base results of test set using RT network.

Path Selection	Request Packing	Base Results
MinMin/FPF	MNOFF	73.67
	MOFF	75.42
ED MinMin/FPF	MNOFF	73.96
	MOFF	74.63
MinCon/FPF	MNOFF	77.93
	MOFF	78.78
LFL-MinCon/FPF	MNOFF	77.93
	MOFF	78.78
GA		77.71
FPF		74.72

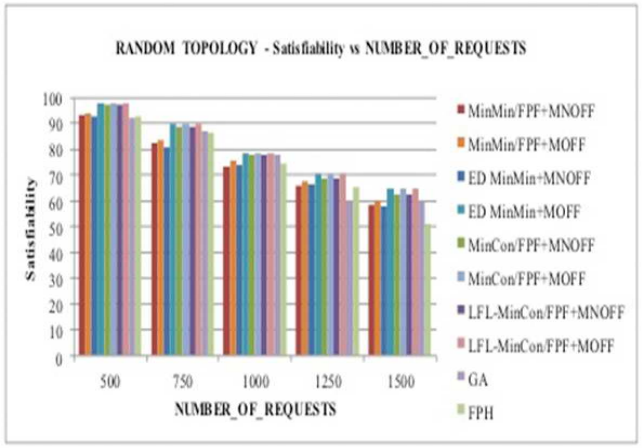

Fig. 11: Effect of varying number of requests on real-time performance in RT network

Performance results of the algorithms are compared against a genetic algorithm solution that is proposed to solve RTU/DDP in this study and a fast and effective method from the literature, namely Full Path Heuristic (FPH), which can solve RTU/DDP.

The algorithms are tested in three different network configurations: GÈANT network, LCG network, and a randomly generated RT network. Considering the real-time performance results and time complexity of the algorithms, it can be concluded best performances are achieved when MinCon/FPF path selection algorithm is followed by MOFF request packing algorithm outperforming the genetic algorithm solution and the FPH.

As a future work, the authors will try to find heuristics to deliver the data transfer requests from multiple sources and from multiple routes.

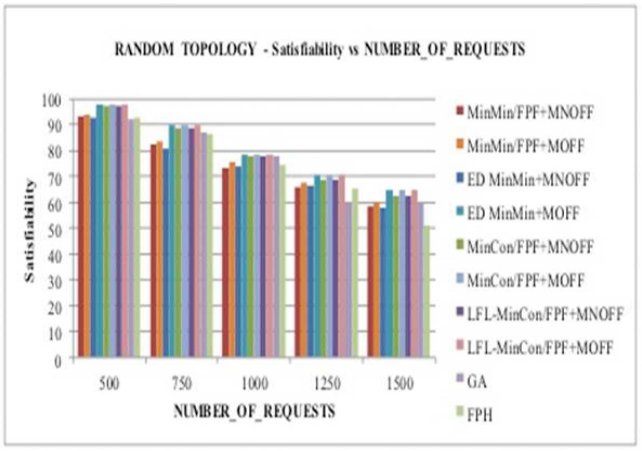

Fig. 12: Effect of varying average request size on real-time performance in RT network

Acknowledgement

This material is based on work supported by Institute of Scientific and Technological Research of Turkiye (TUBITAK) under Grant No. 108E232.

References

[1] The DataGrid Project. Available on: http://eu-datagrid.web.cern.ch/eu-datagrid, June 2014.
[2] D. Bernholdt, S. Bharathi, D. Brown, et al., Proceedings of the IEEE **93**, 485-495 (2005).
[3] Network for Earthquake Engineering Simulation. Available on: http://www.nees.org, June 2014.
[4] C. Nicholson, D. G. Cameron, A. T. Doyle, et al., Journal of Concurrency and Computation: Practice & Experience, **20**, 1259-1271 (2008).
[5] M. D. Theys, M. Tan, N. Beck, et al., IEEE Transactions on Parallel and Distributed Systems **11**, 969-988 (2000).
[6] T. Hea, J. A. Stankovic, C. Lub, T. Abdelzahera, International Conference on Distributed Computing Systems, 46-55 (2003).
[7] DSJ De Couto, D Aguayo, J Bicket, R Morris, A high-throughput path metric for multi-hop wireless routing, Wireless Networks, **11**, 419-434 (2005).
[8] I. Foster, C. Kesselman, C. Lee, et al., International Workshop on Quality of Service, 27-36 (1999).
[9] R. K. Ahuja, T. L. Magnanti, J. B. Orlin, Network Flows, Prentice-Hall, Englewood Cliffs, NJ, 1993.
[10] E. W. Dijsktra, Numerische Mathematik **1**, 269-271 (1959).
[11] R. Bellman, Quarterly of Applied Mathematics **16**, 87-90 1958.
[12] L. R. Ford, Network Flow Theory, Technical Report, P-923, Rand Corp., Santa Monica, CA, (1956).
[13] JC Tay, NB Ho, Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems, Computers & Industrial Engineering, Elsevier, **54**, 453-473 (2008).
[14] T. Kormaz and M. Krunz, INFOCOM **2**, 834-843 (2001).
[15] D. W. Shin, E. K. P. Chong, H. J. Siegel, IEEE Workshop on High Performance Switching and Routing, 385-389 (2001).
[16] R. A. Resende, R. M. Oliveira, F. J. L. Padua, et al., 13th International Conference on Telecommunications, (2006).
[17] RFC 1058 - Routing Information Protocol. Available on: http://tools.ietf.org/html/rfc1058, June 2014.
[18] RFC 2328 - OSPF Version 2. Available on: http://tools.ietf.org/html/rfc2328, June 2014.
[19] RFC 4271 - A Border Gateway Protocol 4. Available on: http://tools.ietf.org/html/rfc4271, June 2014.
[20] R. Guerin, A. Orda, A, D. Williams, GLOBECOM, 1903-1908 (1996).
[21] M. C. Yuen, W. Jia, C. C. Cheung, IEEE International Conference on Multimedia and Expo **1**, 217-220 (2004).
[22] E. S. Jung, Y. Li, S. Ranka, S. Sahni, International Symposium on Parallel Architectures, Algorithms, and Networks, 133-138 (2008).
[23] L. Liu and G. Feng, International Journal of Wireless Personal Communications, **41**, 393-405 (2007).
[24] G. Cabrera and C. Toledo, International Conference on Technologies and Applications of Artificial Intelligence, 322-326 (2010).
[25] Y. Yuan and D. Wang, IEEE International Conference on Automation and Logistic, 340-344 (2007).
[26] Y. S. Yen, R. S. Chang, H. C. Chao, IET Communications **2**, 971-981 (2008).
[27] F. S. Dai and A. J. Liu, International Conference on Wireless Communications, Networking and Mobile Computing, 1-4 (2009).
[28] C. W. Ahn and R. S. Ramakrishna, IEEE Transactions on Evolutionary Computing **6**, 566-579 (2002).
[29] M. Munetomo, Y. Takai, Y. Sato, IEEE International Conference on Systems, Man and Cybernetics **3**, 2774-2779 (1998).
[30] M. Munetomo, N. Yamaguchi, K. Akama, S. Sato, Congress on Evolutionary Computing **2**, 1236-1243 (2001).
[31] M. C. Sinclair, IEEE Conference on Telecommunications, 66-71 (1998).
[32] N. Shimamoto, A. Hiramatsu, K. Yamasaki, IEEE Conference in Neural Networks **2**, 1123-1128 (1993).
[33] M. Hamdan and M. E. El-Hawary, Canadian Conference on Electrical and Computer Engineering **2**, 823-827 (2002).
[34] Z. Yu, Z. Xin, Y. Qingwei, International Conference on Information Science and Engineering, 117-120 (2009).
[35] Y. Leung, G. Li, Z. B. Xu, IEEE Transactions on Evolutionary Computation **2**, 150-161 (1998).
[36] Z. Xiawei, C. Changjia, Z. Gang, International Conference on Communication Technology **2**, 1248-1253 (2000).
[37] Q. Zhang and Y. W. Leung, IEEE Transactions on Evolutionary Computation **3**, 53-62 (1999).
[38] R. J. Al-Ali, K. Amin, G. von Laszewski, et al., Journal of Grid Computing **2**, 163-182 (2004).
[39] RFC 1633 - Integrated Services in the Internet Architecture: an Overview. Available on: http://tools.ietf.org/html/rfc1633, June 2014.
[40] RFC 2212 - Specification of Guranteed Quality of Service. Available on:http://tools.ietf.org/html/rfc2212, June 2014.
[41] RFC 2205 - Resource Reservation Protocol (RSVP). Available on: http://tools.ietf.org/html/rfc2205, June 2014.

[42] R. Callon, J. Jeffords, H. Sandick, J. M. Halpern, Issues and Approaches for Integrated PNNI, ATM Forum, 1996.
[43] RFC - 2676, QoS Routing Mechanisms and OSPF Extensions. Available on: http://tools.ietf.org/html/rfc2676, June 2014.
[44] R. Guerin and A. Orda, IEEE/ACM Transactions on Networking **7**, 350-364 (1999).
[45] Q. Ma and P. Steenkiste, International Workshop on Quality of Service, 115-126 (1997),
[46] P. Goyal and G. Hjalmtysson, International Workshop in Network and Operating System Support for Digital Audio and Video, (1999).
[47] Z. Wang and J. Crowcroft, IEEE Journal on Selected Areas in Communication **14**, 1228-1234, 1996.
[48] W. Grimmell and N. Rao, International Journal on Foundations of Computer Science **14**, 503-523, (2003).
[49] Y. Yang, J. K. Muppala, S. T. Chanson, International Conference on Network Protocols, 62-70 (2001).
[50] S. Chen and K. Nahrstedt, IEEE Network **12**, 64-79 (2001).
[51] J. Kleinberg, Approximation Algorithms for Disjoint Paths Problems, PhD Thesis, MIT, 1996.
[52] M. Eltayeb, A. Doğan, F. Özgüner, IEEE Transactions on Parallel and Distributed Computing **17**, 1348-1359 (2006).
[53] M. Andrews and L. Zhang, SIAM Journal on Computing **37**, 112-131 (2007).
[54] R. Guerin and A. Orda, IEEE/ACM Transactions on Networking **10**, 613-620 (2002).
[55] A. Freville, European Journal of Operational Research **155**, 1-21 (2004).
[56] E. Hernandez-Orallo and J. Vila-Carbo, Journal of Computer Communications **31**, 1218-1226 (2008).

M. Müjdat Atanak received his B.S. and M.S. degrees in electrical engineering from University of Southern California in 2002 and 2004, respectively, and his Ph.D. degree in electrical and electronics engineering from Anadolu University in 2012. He is currently assistant professor in the Department of Computer Engineering, Dumlupınar University. His current research interests include grid computing, embedded systems, and mobile programming.

Atakan Doğan received his B.S. and M.S. degrees in electrical and electronics engineering from Anadolu University in 1995 and 1997, respectively, and his Ph.D. degree in electrical engineering from The Ohio State University in 2001. After receiving his Ph.D. degree, he worked as a post-doctoral researcher at The Ohio State University for nine months. He, then, joined the faculty at the Department of Electrical and Electronics Engineering, Anadolu University in 2002, where he is currently an associate professor. Furthermore, he worked as Deputy Director of Anadolu University Computer Center for three years. Atakan Doğan served as a supervisor or researcher in several national projects, as a researcher in a NFS funded project, and as an advisor for a few national private companies. His current research interests include cloud computing, grid computing, parallel computer architecture, embedded systems, reconfigurable computing, hardware acceleration, and FPGAs.

Mustafa Bayram received his Bis a full professor of mathematics at Yildiz Technical University. Currently he acts as the dean of the faculty of chemical and metallurgical engineering. His research interests include applied mathematics, solutions of differential equations, enzyme kinetics and mathematical biology. He raised many masters and Ph.D. students and is the author of many efficient research articles at prestigious research journals. He is the chief and founder editor of new trends in mathematical sciences journal. He also serves as an editor and reviewer for many outstanding mathematics journals.